IN THE CLAIMS:

1. (Currently Amended) A computer implemented method of allocating stack memory for a process for executing a computer program code, the method comprising:

mapping an active session to a thread for execution, the thread having a first stack memory selected to execute a first class of code associated with a first pre-determined stack memory requirement, the first stack memory corresponding to the pre-determined stack memory requirement;

responsive to a code segment of the code being of the first class, executing the code segment with the first stack memory; and

responsive to the code segment being of a second class of code associated with a second pre-determined stack memory requirement that is greater than the first stack memory requirement, executing the code segment in an auxiliary stack memory corresponding to the second stack memory requirement to execute the code segment, ~~segment;~~ and reclaiming the auxiliary stack memory subsequent to executing the code segment.

a3

2. (Original) The method of claim 1, wherein the code segment includes a function call and code segments of the second class include a wrapper configured to call the auxiliary stack memory to execute the function call.

3. (Original) The method of Claim 2, wherein the thread is non-preemptive, the auxiliary stack memory is a shared stack, and the wrapper performs the operations of:

saving a stack pointer to the first stack;
resetting the stack pointer to the shared stack;
copying arguments from the first stack to the shared stack;
calling a program function of the function call;
returning the result to the first stack of the thread; and
returning the shared stack.

4. (Original) The method of claim 2, wherein the thread is preemptive, the auxiliary stack is a new stack from a pool of stacks, and the wrapper performs the operations of:

saving a stack pointer to the first stack memory;

allocating a new stack segment having a stack address;

saving the stack address of the new stack segment;

resetting the stack pointer to the new stack segment;

copying an argument from the first stack to the new stack;

calling a program function of the function call;

returning the result of the program function to the first stack memory; and

returning the new stack segment.

5. (Original) The method of claim 1, further comprising: allocating a preselected stack memory space for the auxiliary stack memory.

a3

6. (Original) The method of Claim 1, further comprising: allocating the stack memory for the auxiliary stack memory space as required to satisfy the stack memory requirements of the function call.

7. (Original) The method of claim 1, wherein each of the classes includes a code type that is blockable and a code type that is non-blockable.

8. (Original) The method of Claim 7, wherein the code types are identified by a naming convention.

9. (Currently Amended) A method of reducing stack memory resources in a computer system that executes concurrent user sessions, the method comprising:

mapping an active session having a program code to a thread for execution, the thread having a first stack memory space allocated to the thread selected to handle a first class of function calls, the first class of function calls associated with a first pre-determined stack memory requirement, the first stack memory space corresponding to the first pre-determined stack memory requirement;

transferring the execution of the program code from the first stack memory space to an auxiliary stack memory space having a stack memory size greater than the first stack memory responsive to the program code invoking a function call of a second class of function calls associated with a second pre-determined stack memory requirement that is greater than the first stack memory requirement ~~that requires a stack memory size greater than that of the first stack memory~~;

executing the function call on the auxiliary stack memory space, the auxiliary stack memory space corresponding to the second pre-determined stack memory requirement;

copying a result of the function call to the first stack memory of the thread; and

reclaiming the auxiliary stack memory space.

10. (Original) The method of Claim 9, wherein the auxiliary stack memory is a stack selected from a pool of stacks residing in the memory pool.

a3

11. (Original) The method of Claim 9, wherein the auxiliary stack memory is a shared stack.

12. (Original) The method of Claim 9, further comprising: selecting the size of the auxiliary stack memory as a function of a code type of the function call.

13. (Original) The method of Claim 9, further comprising: wrapping the program code in a wrapper to transfer the execution to the auxiliary stack memory.

14. (Original) The method of Claim 13, wherein the thread is non-pre-emptive and the wrapper performs the steps of:

saving a stack pointer to the first stack memory;

resetting the stack pointer to a shared stack;

copying arguments from the first stack to shared stack;

calling a function;

returning the result of the function to the first stack; and

returning the shared stack.

15. (Original) The method of Claim 13, wherein the thread is pre-emptive and the wrapper performs the steps of:

saving a stack pointer to the first stack;
allocating a new stack segment having a stack address;
saving the stack address of the new stack segment;
resetting the stack pointer to the new stack segment;
copying the argument from the first stack to the new stack;
calling a function;
returning the result of the function to the first stack memory; and
reclaiming the new stack segment.

16. (Original) The method of Claim 9, wherein the first class includes a code type that blocks and a code type that does not block

a3

17. (Original) The method of Claim 9, wherein the second class of functions includes a code type that blocks and a code type that does not block.

18. (Currently Amended) A method of programming a computer program user code for execution by a thread with a default stack memory in a threaded computer system, the method comprising:

prior to execution of the function calls of the program code, identifying function calls of the program code requiring stack memory greater than the default stack memory allocated to the thread; and

wrapping each function call requiring stack memory greater than the default stack memory ~~that~~ allocated to the thread with a wrapper configured to call an auxiliary stack memory to execute the function call.

19. (Original) The method of Claim 18, further comprising:

selecting the stack memory allocated to the thread sufficient to handle a first class of function calls.

20. (Original) The method of Claim 19, further comprising the step of: selecting the size of the auxiliary stack memory sufficient to handle a second class of function calls.

21. (Original) The method of Claim 18, wherein the auxiliary stack memory is a new stack from a memory pool.

22. (Original) The method of Claim 18, further comprising the step of:
forming a shared stack as the auxiliary stack memory.

23. (Original) The method of Claim 18, wherein the code includes a function call having a recursive algorithm, further comprising:
replacing the recursive algorithm with an iterative algorithm performing the same function, whereby the size of the stack required to execute the function is reduced.

a3

24. (Original) The method of Claim 18, wherein the function call includes a stack-allocated variable and further comprising:
replacing the stack allocated variable with a heap allocated variable, whereby the size of the stack required to execute the function is reduced.

25. (Original) The method of Claim 18, further comprising:
identifying a program code segment that blocks substantially longer than other program segments; and
replacing the program code segment with program code segment(s) performing the same function but selected to reduce the potential blockage time.

26. (Original) The method of Claim 25, wherein a supervisory program having a database of program code segments is used to implement the method.

27. (Original) The method of Claim 18, wherein each function call has a corresponding program code naming convention.

28. (Original) The method of Claim 18, wherein the program code is executed in a program language having checked exceptions and the different classes of code are declared to throw different classed of checked exceptions.

29. (Original) The method of Claim 18, further comprising the steps of:
classifying different type of function calls into a classification based upon stack memory usage;
preparing a database of wrapper functions, each wrapper function associated with a type of function call to implement the function call as a wrapped function calling the auxiliary stack memory; and
assigning a wrapper to each function call based upon the classification.

30. (Original) The method of Claim 28, wherein a computer assigns the wrapper.

a3

31. (Original) The method of Claim 18, further comprising the step of:
characterizing at least one function by running the function on a real or virtual system to determine the stack memory required to execute the function.

32. (Original) The method of Claim 19, further comprising:
characterizing at least one function call by running the function on a real or virtual system to determine if the function is blocking or non-blocking.

33. (Currently Amended) A computer readable medium including program code for execution of a process in a computer system, the computer system having at least one computer thread having a first stack memory having a first stack size allocated to the thread and an alternate stack memory space having a second stack size, the program code comprising:
a computer program code having code segments of different code class, the code including a first ~~code~~ class of code associated with a first pre-determined stack memory requirement of a ~~that requires the~~ first stack memory size and a second ~~code~~ class of code associated with a second pre-determined stack memory requirement of a ~~that requires the~~ second stack memory size; and

a wrapper wrapping each code segment of the second class configured to transfer execution of the function to the alternate stack memory space.

34. (Currently Amended) A computer system having an operating system for concurrently executing a plurality of user session requests, comprising:

a computer program residing in a memory, comprising:

a pool of threads, each thread having an associated stack memory having a first stack size;

a thread mapper mapping each user session onto one of the threads in the pool of threads;

an auxiliary stack memory having a second stack size, the second stack size being larger than the first stack size;

a program code for executing one of the user sessions, the code including at least one code segment characterized by a code class, the code classes including a first class of code ~~class~~ associated with a first pre-determined stack memory requirement corresponding to the first stack memory ~~that requires the first stack memory size~~ and a second class of code ~~class~~ associated with a second pre-determined stack memory requirement corresponding to the auxiliary stack memory ~~that requires the second stack memory size~~; and

a wrapper for each code segment of the second class configured to transfer execution of the function of each code segment of the second class to the auxiliary stack memory.

a3

35. (Currently Amended) A computer thread for executing program code, comprising:

a first stack memory associated with the thread for executing a first class of function calls associated with a first pre-determined stack memory requirement of ~~requiring~~ a first stack memory size; and

switchable auxiliary stack memory means for executing ~~function calls of~~ a second class of function calls associated with a second pre-determined stack memory requirement~~requiring a stack memory~~ resource greater than the first stack memory and reclaiming the stack memory resource when the function call of the second class is completed.